(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 October 2001 (25.10.2001)

PCT

(10) International Publication Number
WO 01/80098 A2

(51) International Patent Classification⁷: G06F 17/30

(21) International Application Number: PCT/US01/40517

(22) International Filing Date: 13 April 2001 (13.04.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/197,140 14 April 2000 (14.04.2000) US

(71) Applicant *(for all designated States except US)*: **MAYIM LLC** [US/US]; 1205 De La Vina Street, Santa Barbara, CA 93101 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **MARTIN, Patrick** [US/US]; 561 La Marina Drive, Santa Barbara, CA 93109 (US). **MOHLER, Mark, D.** [US/US]; 1970 Mission Ridge Road, Santa Barbara, CA 93103 (US). **BROCK, Phil** [US/US]; 407 La Marina Drive, Santa Barbara, CA 93109 (US). **PARKS, Michael** [US/US]; 807 Arguello, Santa Barbara, CA 93103 (US). **RHODES, Matt** [US/US]; 729 Juanita Avenue, Santa Barbara, CA 93109 (US).

(74) Agents: **DEVAUL, William, D.** et al.; Foley, Hoag & Eliot, LLP, Patent Group, One Post Office Square, Boston, MA 02109 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

WO 01/80098 A2

(54) Title: WEB BROWSER PLUG-IN PROVIDING 3D VISUALIZATION

(57) Abstract: Methods, systems and means for providing a plug-in for a web browser that enables a host or content provider to deliver a 3D visualization environment to an end user.

Web Browser Plug-In Providing 3D Visualization

Background

1. Field of the Invention

This disclosure relates to the field of the digital delivery of content, and more particularly to the field of delivery of 3D visualization content over computer networks.

2. Description of the Related Art.

The advent of computer networks, particularly the Internet, has enabled a wide range of new commercial interactions, including online shopping.

Current methods and systems for presenting material to online users have a number of problems. The predominant methods remains presentation of static content, in the form of HTML pages, that are viewed by a user employing a web browser. The pages are similar to pages of a magazine, containing text, pictures, and the like.

For more dynamic content, a variety of other problems exist. One problem is limited bandwidth in computer networks. Thus, applications such as streaming video and audio are inherently limited by the ability of the computer network to deliver content at a rate that provides high quality video or audio. Thus, a variety of compression techniques have evolved to reduce the size of a file while retaining acceptably high quality in the transmitted content. Nevertheless, video and audio content continues to be hindered by large file sizes and limited bandwidth. Thus, users typically have to wait a significant amount of time to receive such content.

Another problem with existing dynamic content is its lack of interactivity. That is, even dynamic content, such as streaming audio or video, typically provides the user with minimal opportunity to interact with the content. A user may, for example, select a portion of video to play, play the video, stop the playing, and perhaps change the volume. However, the user is very limited in terms of other interactions with the video, such as changing the user's point of view.

A few systems have emerged that permit some viewing of different views in video via the Internet. For example, there are existing systems that allow a user to view a rotating object, such as a model or human figure, using a series of views that are created by positioning cameras around the object and by linking the resulting photographs. Other systems allow a user to perform navigation through an environment, such as a home, by showing the user pictures of the home and allowing the user to navigate using the mouse or keyboard to different points. Again, the environment is created by aggregating photographs taken from particular points of view.

Existing Internet 3D visualization programs are limited in that they do not permit the user to have free navigation within the environment that is created. This is because they are limited to the points of view of the cameras that are used to take the photographs or video that are used to create the view. Thus, a user can navigate to certain points, but is not given an ability to freely navigate throughout the entirety of an environment.

Certain video games give the user the ability to navigate within a 3D visualization environment, but such games are typically created with proprietary software for proprietary hardware platforms, thus making them ineffective for use in a client-server environment such as the Internet.

A need exists for methods and systems that provide an enhanced user experience, including real-time interactivity with environments visualized in three dimensions, that can be delivered to users who are using the Internet.

Summary

Provided herein is an interactive 3D visualization plug-in for a web browser. Once a user installs the plug-in, the user can navigate through a wide range of 3D visualization environments using a conventional web browser. The environments may be served to the user's browser by one or more servers that are connected to one or more databases that store data that is used to build the environments.

Detailed Description

Provided herein is an interactive 3D visualization plug-in for web browsers. It allows a web page to contain a 3D viewport driven by, for example, either OpenGL or Direct3D. The viewport both displays 3D information contained in a "scene" and manages user-driven interaction via input devices (e.g. mouse, keyboard) and on-screen navigation tools (also within the viewport).

Included herein is a downloadable plug-in for browsers, such as Netscape Navigator and Internet Explorer, which uses streaming 3D technology to transmit 3D information from a server (typically remote) to a browser enabled device (typically a desktop PC or laptop, but possibly any device with a graphical interface that is browser-enabled). A browser user with the plug-in installed may download, walk through, and interact with a 3D visualization environment. Certain features of the end user's interactive experience include real-time visualization of geometry with color, texture, and shading, real-time navigation within the environment, alternative site map display (dynamically built) for global positioning/location, rigid-body animation, object selection, and pre-programmed actions (e.g. navigation, animation, and HTML functionality) triggered by object selection.

The systems, methods and means disclosed herein may be useful in an configuration that includes a host, who may be content provider or a company that provides 3D visualization services to a content provider, one or more content providers, who may, in an embodiment, be providers of commercial, retail, or marketing content, such as online stores, online malls, catalogs, retail goods and services, advertisements, offers, online tours, video samples, promotional games, and the like, and one or more end users. A configuration for this environment may include an end user device, which may be any device capable of connection to a computer network, such as the Internet, and capable of receiving and executing computer programs or files through, for example, the Internet. The user device also preferably includes an Internet browser, such as Microsoft Internet Explorer or Netscape Navigator, that is capable of displaying graphical content. In embodiments, the user may connect to the computer network, such as the Internet, to receive electronic mail or to receive files through any other mechanism, such as file transfer protocol, or by downloading from an Internet site.

The host system may include one or more servers that are enabled to provide content over the Internet, including the Worldwide Web, such as an Apache web server, UNIX machine, Compaq machine or other web server. The server (which may be a hardware server, software server, server process or other method or system for serving files or other data over a computer network) preferably includes various other components, or is associated with a computer system that includes such components, including an operating system, an input device, and one or more databases. The databases may store content, such as data files that contain data to support three dimensional environments, objects, scenes, models, and the like. It should be understood that as used herein the terms scene, environment, model, and object are intended to encompass any item that is capable of graphical representation in a 3D visualization environment.

A third party content provider may interact with the host through any of a variety of conventional methods and systems, including by providing content to a database that is accessed by the host through the Internet, by providing content directly to the host, such as through electronic mail, FTP, or other online mechanism, by providing content directly to the end user via tools that are provided by the host, or by land-based systems, such as diskettes, compact diskettes, digital video disks, tape files, or the like. The content provider and the host may interact to determine what content, e.g., a three dimensional environment, is desired to be provided to an end user. In embodiment, the interactive tools can access multiple distributed databases on the fly and can create three dimensional visual environments or the end user in real time.

Provided herein are a plurality of 3D visualization tools that assist the host in providing highly effective content to end users. Key elements of these tools are their ability to provide real time interactivity for end users in 3D visualization environments, their ability to provide rich three-dimensional visualization content in real time, even with limited bandwidth, and their ability to integrate electronic commerce transactional systems with three-dimensional visualization environments. Other elements and benefits will be readily understood by those of ordinary skill in the art, or are described further herein.

Two key benefits of the methods and systems disclosed herein are improved user navigation and prioritized progressive refinement.

A user interacting with an environment displayed and managed by the plug-in application disclosed herein will be able to visualize an environment or scene through a single viewport. The contents of the scene can be viewed from multiple points of view, including the user's point of view (their current position and orientation within the scene), one or more "sitemap" points of view, or other points of view chosen by the host or content provider. The sitemap, for example, can be displayed as either an exploded perspective view, a more traditional orthogonal view, much like a map, or some other selected point of view. The sitemap can show the user's position within the environment or scene and any defined anchor points (e.g. pre-defined vantage points). A scene database can be designed to contain scenes that are digitally constructed such that each "building" (generically an object) has a representation for any desired point of view that the host or content provider selects, such as the user's point of view and the sitemap. This allows multiple points of view to be uniquely defined and supporting from the same methodology and data set.

This form of navigation - and the ability to walk through a virtual environment and quickly switch back and forth between your current point of view and a sitemap (much like walking around Paris, France with a guidebook in hand) – provides significant advantages relative to a conventional site map, both from a user's perspective and from a web-site management perspective. Instead of having to create a separate map of a virtual environment, the map is created automatically from the same data as the scene.

In embodiments, the plug-in applications described herein can provide a site map that is similar to a guidebook within the user's point of view display. This view, could, for example, incorporate a pair of digital hands lifting the guidebook to eye level for the user.

Another significant advantage provided by the methods and systems disclosed herein is prioritized progressive refinement. In embodiments, a compiled scene database (WCF) that stores scene data for the methods and systems disclosed herein can allow for a large flexibility in how scene or environmental data gets downloaded to a client. The WCF (explained in detail below) can contain a blueprint outlining the complete scene description. Logic within the client-side plug-in can query the scene database in order to get what data should be downloaded next from the server. This logic looks at the blueprint and the current user position and determines a prioritized list of scene information that should be downloaded to the client to make the display of the scene qualitatively better.

This whole technique can be termed Prioritized Progressive Refinement (PPR). After the client receives the blueprint, the scene can be initially displayed and then continually updated (progressively refined) with information that provides greater and greater levels of detail. The updating is done using a prioritized scheme that includes both static and dynamic prioritization. Static prioritization is derived from information in the WAF file (a format described below) and information gleaned during the compilation of the WAF file into a WCF file as described below. These static priorities are encoded in the WCF. Dynamic prioritization occurs on the client side when comparing the blueprint to the current user position and orientation. A final download priority for a piece of scene information is then computed from a non-linear weighting of the static and dynamic priorities. PPR can be enabled by database structures that are analogous to the layers of an onion, with each additional layer of data in the database relating to an additional layer of detail that is used to populate a scene. For example, the first, or outer layer of a data structure might contain the basic geometry of objects in a scene, such as rectangular solids (representing buildings), planes (representing the ground) and the like. The next layer might contain additional details, such as primary color. Other layers may include more detailed colors, textures, and the like.

The WCF can be designed such that a web server can easily find and return (via standard internet communication protocols) individual pieces of a scene. The various scene pieces use varying encryption algorithms based upon scene class type.

In embodiments, 3D visualization environments can be stored on the user's device, such as on a hard disc, for use while the user is offline. Thus, upon executing the executable code of the application, which starts when the user performs the standard execution procedures, such as double clicking on a graphical icon, issuing the run command against the executable, or other procedures, the application automatically initiates a function to determine whether the user is online. If so, then the application polls the server of the host or content provider, and further instructions can be obtained from the server, in combination with code of the executable, as described elsewhere herein. If the user is not online, the processing is handled entirely by the code can be contained in the application. In embodiments, it would be possible to store code on the user's computer or another server.

When the user is interacting with the server, it is possible to store information about the user's activity in interacting with the application, by tracking the clickstream, storing a cookie, or the like. When the user is off-line, this data is not stored on the server. Therefore, in embodiments, the user's activities can be written to a file, such as a hidden file, which can be accessed and uploaded (automatically), the next time the user is online. Thus, an instruction to query whether the user was previously offline can be included in the initial execution routine.

The methods and systems disclosed herein are intended to work in a typical web-server environment. A browser plug-in (client tier) communicates with one or more web servers that return requested scene information. The browser plug-in can be a modularly designed application written in C++. It may contain a number of principal modules, including one for sending and receiving requests, one for displaying, and one for user navigation and selecting. These three modules are termed herein the cache, the drawer, and the manipulator. These three modules can be threads in a conventional Windows application. The application's main loop handles all mouse, keyboard, and time based activities.

Client/server communication (including streaming 3D) can be performed by standard Internet protocols, such as TCP/IP and HTTP. For streaming 3D, packets sent from the server to the client can be scene item based. Any object requiring data exceeding a certain size, such as

10KB, can be broken into more than one packet in a consistent manner, such that a request for the x'th packet of an item will always result in the same information being sent.

Referring to Fig. 1, the methods and systems disclosed herein can include a scene database that can be used with the plug-in. Creation of the scene database can involve a multi-step process, such as a two-step process. A first step is to model the environment. Modeling can be accomplished either with a proprietary modeling system, or with a conventional 3D software modeling package, such as LightWave, Maya, Softimage, or 3D Studio Max. The package can be provided with a translator, such as an ASCII FORMAT Translator, referred to herein as a Walkabout Ascii Format (WAF) translator (which may be a plug-in). Once a scene has been modeled and saved in the WAF format, it can be compiled using a scene compiler into a more efficient format, referred to herein as the Walkabout Compiled Format (WCF).

The WAF can be an ASCII-based file format that in the least can be edited using any standard text editors. The WCF can be a compact, encrypted binary format. The scene compiler can actually produce two WCF files for each scene, a small header file and a file containing the complete scene description. The WCF header file can be initially read by a browser and can contain information about where the actual scene information is found and how to request it.

The methods and systems disclosed herein can be provided in a multi-tier client-server architecture, where a client device (browser enabled, or enabled with other graphical interfaces, such as WebTV) connects to a server, which may direct various second-tier functions, such as accessing database servers, accessing a plug-in server, load balancing, or the like. The client side device may have various modules, including a module for sending requests over a computer network, a module (such as a browser) for receiving, parsing, and handling message from the computer network, a drawing engine, which may be connected to a database that contains scene data that is stored on the client device, and a user navigation module, used to handle user navigation in the environment drawn by the drawing engine. The client-side database may have a multi-layer construction, such that certain data is accessed in an appropriate sequence to populate a scene in a desirable order (thus enhancing speed of use). For example, data relating to items in close proximity to the user's point of view may be stored for rapid retrieval when the user reenters a scene, while minor details, such as textures of distant objects, may be retrieved and drawn only if the user is in the scene for a longer period of time. Databases may be partitioned to contain a variety of objects, which may be stored separately within the databases, or in different databases. The primary server of the host system may serve various data from its own database in response to requests, which may be based on client-side navigation. For example, if a client rests a mouse over a particular object, the host server may identify in a scene database additional data for the object (such as data needed to rotate the object in three dimensions) and send the data to the client side for display to the user. The host server can also recognize first-time users, so that they can be provided with the plug-in for initial installation. Other items that can be included in the database include any kind of information that can be visually or audibly represented, such as URLs, text, video, audio, graphics, photographs, drawings, colors, textures, and the like. Any of these items may be presented in a 3D visualization environment. For example, a picture or texture can be wrapped about an object, such as a cube or sphere. For example, a conventional banner advertisement might be wrapped around a sphere to produce a more interesting visual effect, calling attention to the banner ad.

The client side database can also be populated while the client is connected to the site, including with data that relates to scenes that the client is not currently viewing. The application can also load balance between downloading and sleeping, based on activity levels of other applications on the client side machine. Each of the client side modules may be a C++ module or may be written in other object-oriented languages such as JAVA.

Currently available, off-the-shelf 3D systems can be used to model geometry, to create and apply materials and textures, and to create animation information. In embodiments, a single texture may be applied per object. In other embodiments, multiple textures may be overlaid to provide combined texture effects. A translator plug-in for any one 3D system can exist for *reading and writing a WAF file.*

Slightly different versions of the plug-in can be created for different operating systems. For example, in a Windows version, the application can be configured to respond to events, such as mouse, keyboard, or draw events, as in standard Windows applications. Similar programming is used for the Macintosh or LINUX operating systems, for example, but with different programming syntax. In each case, standard operating system APIs are used to track mouse and keyboard events. In Windows, draw events are timed, so that a draw event occurs every x milliseconds, resulting in drawing of a scene. With the MacIntosh operating system, rather than providing a timer, the application may simply draw a scene whenever there is no other event (i.e., a null event occurs). Similar or identical data structures and drawing routines may be used for the various different operating systems.

The plug-in can be delivered and installed by various mechanisms, such as manual installation, semiautomatic, and automatic mechanisms. In an embodiment, the plug-in, once downloaded, installs itself automatically. The initial download may be a single file, which, when executed, creates other files. In embodiments, the file may be a compiled dynamic link library, or DLL (in Windows) or library (for other operating systems). The DLLs or libraries are binary files that have external links to them that other programs can call. The systems and methods disclosed herein thus include a client-side module used as a plug-in, which can be plugged into another program, such as the user's browser. For example, the browser, e.g., Netscape, automatically looks in plug-in folder on the user's computer, keeps track of the various plug-ins in the folder, and calls in the plug-in whenever it finds a program that uses that plug-in. Thus, the plug-in attaches itself to the program. Thus, the client-side of the present methods and systems consists of a custom DLL or plug-in for any conventional web browser. Different versions of the plug-in can exist for different client operating systems. Although there may be a different plug-in for each platform, the remaining code may be consistent across different operating systems. Thus, the data structures and 3D visualization software can be consistent, regardless of the client operating system. In embodiments, the core code may be programmed in an object-oriented language, such as C++.

A variety of different environments can be presented to an end user. Thus, a scene can be described for use as multiple displays, such as both an immersive display and as a site map display. An immersive display occurs from the user's point of view. A site map display occurs from a traditional top view where the camera is placed above (in Y) looking down the negative Y axis using either an orthographic or exploded perspective view. For applications such as virtual malls, store interiors, etc., it is advantageous for buildings to contain a roof which is not seen during the immersive display, but which is seen during the site map display. This way, maintenance of scenes which may be continually modified over time (e.g. stores added) will not require changes or modifications to a separate site map.

Further details as to the Walkabout ASCII Format include the following. Positions may be defined in world space, e.g., a Cartesian coordinate system for a space, with coordinate systems for particular objects. Directions and angles may be provided in degrees. Colors may include a default color, such as black. Transparency characteristics can be provided, ranging in opacity. The default opacity can be set to opaque (1.0). Inputs are preferable case insensitive, except for text in quotes. Normal coordinates may be set with defaults, such as (0,0,1). Texture vertices can be set with defaults, such as (0,0). If you look at a square, and want to map an image on that, you have to map which color of a texture goes onto which corner of the square. A texture is an image that is wrapped around an object. The texture can be any image. There can be particular format. Materials can be set with default characteristics, such as diffuse-only (0.8,0.8,0.8). Transforms can default to translate: (0,0,0) rotate: (0,0,0) scale: (1,1,1). If a geometry does not have polygons, then the point data can be rendered as a point cloud. The default "camera lens" for the 3D visualization can be set at a particular focal length, such as 50mm. File specifications can use specified formats, such as "frame.####.ext" or frame.@.ext format. The default priority can be set at an arbitrary number, such as 50. Key words can be established and distinguished, such as by placing them in bold.

An example of a WAF file is as follows:

```
begin Scene "<name>"

        [background color <r> <g> <b>]
        [background texture "<name>"]
        [sitemap <yscale> <lense>]

        begin Material "<name>"
                [ambient       <r> <g> <b>]
                {diffuse       <r> <g> <b>]
                [specular      <sharpness> <brightness> ]
                [transparency<t>]
                shade_model {A][D][S][T]
        end Material

        begin Light "<name>"
                type           spot|point|sun|ambient
                color          <r> <g> <b>
                [[transform    "<name>"] |
                 [[translate   <x> <y> <z>]
                  [rotate      <x> <y> <z>]]]
                [cone_angle    <angle>]
                [attenuation   <atten>]
```

```
end Light


begin Geometry "<name>"
          [type          default|camera_facing]
          point          <x> <y> <z>
          ...


          [normal        <x> <y> <z>]
          ...


          [texture       <u> <v>]
          ...


          [poly3         <p1>/[<n1>]/[<t1>] <p2>/[<n2>]/[<t2>] <p3>/[<n3>]/[<t3>]]
          [poly4         <p1>/[<n1>]/[<t1>] <p2>/[<n2>]/[<t2>] <p3>/[<n3>]/[<t3>]
<p4>/[<n4>]/[<t4>]]
end Geometry


begin Texture "<name>"
          filename       "<file path>|<file specification>"
          [frame_start   <integer = frame number>]
          [frame_end     <integer = frame number>]
          [interval      <integer = milliseconds>]
```

```
        [loop          <true|false>]
end Texture


begin Object "<name>"
        geometry       "<name>"
        [material      "<name>"]
        [texture       "<name>"]
        [priority      <integer, (lowest) 1-100 (highest)>]
        [[transform    "<name>"] |
         [[translate   <x> <y> <z>]
          [rotate      <x> <y> <z>]
          [scale       <x> <y> <z>]]]
        [highlight     "<action description>"]
        [select        "<action description>"]
end Object
```

"<action description>" = "*function*(*[arg1,...,argn]*);[*function*([*arg1,...,argn*]);]..."

```
begin Transform "<name>"
        [translate     <x> <y> <z>]
        [rotate        <x> <y> <z>]
        [scale         <x> <y> <z>]
        [animType      absolute|relative]
```

```
                                        [animLoop     <true|false>]
                      [keyframe     <time, in milliseconds> <type = t|r|s> <x> <y> <z>
                                      <interpolation method = constant|linear|spline]
                                                  ...
    end Transform

    begin Group "<name>"
         [[transform    "<name>"] |
          [[translate   <x> <y> <z>]
           [rotate      <x> <y> <z>]
           [scale       <x> <y> <z>]]]
         [highlight     "<action description>"]
         [select        "<action description>"]
         [child         "<object name>"]
         ...
         [child         "<light name>"]
         ...
         [child         "<group name>"]
         ...
    end Group

    begin Camera "<name>"
         anchor         "<name>"
```

```
            lens            <value = lens size (mm)>
        end Camera


        begin Anchor "<name>"
            translate       <x> <y> <z>
            rotate          <x> <y> <z>
        end Anchor


    end Scene
```

The compiled form of the scene description differs radically from the Ascii format. Whereas the text form of a scene is represented by a directed acyclic graph (DAG), the compiled form is based on a series of lists. The WAF contains named objects, lights, etc. The WCF contains names and id's (integer counters/indices) for each scene element.

The goal of the WCF is to represent the scene in as compact a format as possible. All duplicate data will be compressed into a single form. All extraneous or default data need not be stored.

The WCF may also be encrypted. Each type of scene item will have its own encryption algorithm enabling each scene element to be individually encrypted. Each encryption algorithm will involve some form of byte and/or bit swapping and check summing. It will also be important that older versions of a plug-in be able to read newer versions of a WCF. For this to happen, the number of bytes being written for some data types will be included in the file so unknown data types can be skipped.

A WCF scene downloaded to the plug-in application can be stored within an internal scene database. The design of the scene database is based upon both a directed acyclic graph structure as well as standard arrays. Nodes within the DAG represent various types of scene entities, such as Objects, Lights, Cameras, or Groups. Certain high-level C++ classes that the DAG incorporates are listed below.

ActionList class: A text string. An empty text string equals null. Otherwise can contain one or more actions. Actions are differentiated between browser actions (open url, fill frame with url) and 3d actions (open scene, transport user, goto anchor, run animation, show description, glow).

Material class: Contains RGB_Ambient, RGB_Diffuse, RGB_Specular, A_Transparency, Bits_Shadingmodel.

Texture class: A filename and/or file specification, with possibly some animation related information for animated textures.

Geometry class: name (for scene compiler), index (for plug-in), bounding box, and lists for polygons, points, normals, and texture vertices. Polygons will be able to be grouped as either fans, strips, or individuals.

Transform class: XYZRotation, XYZTranslation, XYZScale, optional keyframes [header + keyframes], animation applied (relative vs. absolute).

Object class: Geometry reference, Material reference, Texture reference, priority, optional highlight Action reference, optional select Action reference, parent Group reference.

Light class: type, parameters, Transform, optional parent_id.

Group class: list of child SceneMember (Object, Light, Group) references, optional Transform reference, optional highlight ActionList reference, optional select ActionList reference, bounding box, optional parent Group reference.

Camera class: type, parameters, parent Group reference.

Anchor class: name, position, orientation.

Scene class: group list, object list, light list, geometry list, material list, texture list, anchor list, transform list, actionList list, header information. Header information also contains such info as optional site map. The group and object lists are organized such that all "hot" items are ordered first, followed by "static" items. Hot items are those which have actions associated with them and which must be tested for various conditions during critical performance functions such as hit testing. Static items are things which have no actions associated with them.

It should be noted that the scene compiler uses this internal scene database too. Within the C++ code, the MAYIMPLUGIN and MAYIMCOMPILER compiler directives are used to filter and reduce class functions by application, resulting in smaller and more efficient applications.

A scene description can be streamed from a database server to a client browser using a technique that is termed herein *prioritized progressive refinement* (PPR). When a client first requests a scene, all the primary scene information is downloaded so that the scene can be quickly displayed. This primary scene information contains, for example, the camera, lights, objects, materials, and groups. After the primary information has been received, the client requests further (detailed) information based upon a prioritization scheme involving such things as the current user position and orientation in the scene (what the user sees) and data type (texture vs. animation). The following types of information can be incrementally downloaded:

- Texture resolution
- Animation data
- Animated textures
- Objects and/or Geometry may be implemented for low bandwidth situations

It should be noted that a part of the primary scene information contains the "blueprint" of the scene. This blueprint has its own time stamp and is used to convey to a client that the scene has been modified on the server and needs to be updated. This allows for a host or content provider hosting an application to perform incremental modifications to the 3D scene without the scene having to be re-downloaded in entirety.

The scene compiler can be a stand-alone application written in C++ that is used to convert a WAF file into a WCF file. The compiler contains a token parser for reading the WAF file into the scene database classes. It then "compiles" the scene into the more compact representation, and writes out the WCF file. The WCF file is actually a directory structure of encrypted binary files. Using object-oriented methodologies, each class knows how to write itself in the WCF format.

When making modifications to an existing WCF file, the compiler can take both a WAF and a WCF file as inputs and create a new WCF that is a revision of the previous WCF file. The compiler allows for adding new elements to a previous WCF, as well as removing or modifying existing elements within a previous WCF.

The optimization procedures used in the compiling of a scene involve a heavy use of instancing. Instancing is a technique whereby different items share common attributes. The common attributes are defined once and referenced, instead of being multiply defined for each usage. The compiler can also push glow actions into the leaves of the DAG, push non-animated transforms down the DAG and consolidate multi-level transforms as much as possible, and organize the group list so all root level groups come first.

The plug-in methods and systems described herein can include a startup or installation routine, a set of threads, such as cache, draw and manipulator threads, methods for obtaining data from a cache, or a server, databases for storing scenes, and methods for providing interaction for the user with the scene, resulting in different displays.

A user can interact with a scene in a variety of distinct ways: navigation, highlighting, and selecting.

All user interaction may be managed by a "manipulator" thread that is given mouse and/or keyboard activities, determines what the proper response is, and calls the appropriate internal functions to handle the specified interaction.

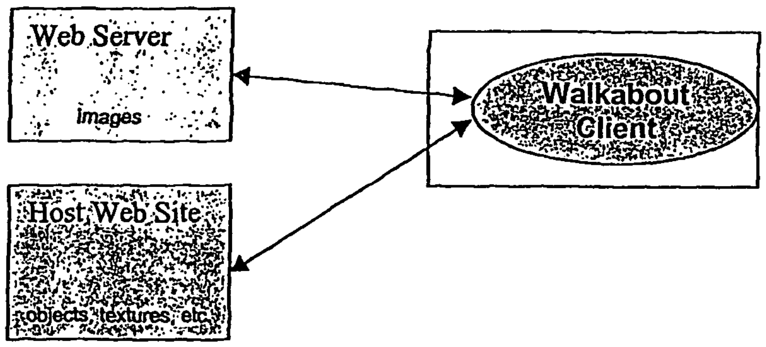
User navigation is performed via the navigation keyboard keys (principally the up, down, left, and right arrows and the page up and down) and/or the mouse buttons which moves the point of view within the 3D visualization environment presented to the user. A change in the point of view can occur from either changing the viewer position or viewer orientation.

User highlighting occurs when the mouse – with no button depressed - is situated over a "hot" object. A hot object is an object with highlight actions defined (or an object whose parent group(s) has highlight actions defined). The mouse is over an object if the object is the closest object to the viewer where the mouse pointer is located. If the hot object is not currently responding to a previous select action, then the highlight actions for the hot object (group) get triggered.

User selecting occurs when, for example, the mouse button is depressed while situated over an object (group) with defined select actions. If the object is currently responding to a previous select action, then the select action is terminated.

The following discloses the scene member caching mechanism, i.e., the caching mechanism used when streaming scene members from a remote server. This portion of the code works in conjunction with the other components of the methods and systems disclosed herein. This set of functionality can exist, for example, in a thread.

To understand how the mechanism works, consider the following:



The client application runs on a user's computer. It retrieves the scene data from a web site containing the scene or environmental data. Since the WAF and WCF formats allow the referencing of images on other web sites, additional servers (which may be geographically distributed) may need to be accessed in order to fully retrieve all portions of the scene. For example, a scene might include video from one site, a picture from another site, advertising displays from another site, and the basic objects of the environment from another site.

The client caching mechanism is responsible for requesting, accepting and relaying data from the remote web sites to the local client. In embodiments, the client may not keep track of the specifics of the scene, in which case it must ask the scene what it needs. Once it has retrieved the requested data, it gives the data buffer to the scene for processing.

The caching mechanism may follow the following procedure. First, it may ask the scene what should be retrieved next. If there is space available in the cache (either because this cache isn't full or because the buffer has higher priority than others, then it may create a buffer large enough for the required object. It may also ask the browser to retrieve the URL related to the object, and it may bump any incomplete data buffers so that the cache doesn't exceed its maximum size. As the data arrives, the caching mechanism may append the data to the end of the related buffer. Once a buffer is full, the mechanism may give the buffer to the scene so that the buffer can be decrypted and parsed. If the user jumps to a new point, such as a new anchor point in the scene, the cache mechanism can clear low-priority buffers (such as based on a priority cut-off). If the user jumps to a new scene, all buffers may be cleared.

Since each scene member can have an associated priority, the site designer can specify which objects will be downloaded and displayed first.

In Windows, the plug-in application can be coded in C++ using run-time libraries, such as the WIN 32 run-time libraries. The environment for creating the executables can be Microsoft Visual C++ or a similar compiler. In a Mac version, the executables can be written in C++ using appropriate run-time libraries for the Mac OS. The design environment can be any suitable design environment for coding 3D visualization environments.

In the case of an automatic download, the server can "sniff" to determine the type of browser and platform. Applications for other platforms can be created.

In embodiments, the tools disclosed herein can be dynamically created 3D visualization environments. For example, if there are multiple possible 3D environments or objects that can be included as content, the host server can identify the profile of a user, know their preferences, or know other items about the user, and could assemble a 3D visualization environment and send it to them based on these factors. Such a system is not limited only to individual user preferences, but to any other data relevant to the content that the host provides; for example, the host might dynamically create a different 3D environment depending on the time of day, month or year, other content in the web environment the user is visiting, where the user came from on the Internet, the user's gender, the user's bandwidth, or the like. Thus, the system can take inputs in real time and build different 3D visualization environments based on the inputs by accessing different data residing on, for example, the host server, any other server, or the user's device.

It should be understood that the user device could be a personal computer, laptop, notebook, PDA, handheld device, pager, cellular or other phone, set top box, WebTV or any other browser-enabled device that has a graphical interface.

In the Windows environment, the application can be a standard Windows application (i.e., it has a standard entry point and exit point defined by the Microsoft runtime libraries, it contains a main loop, it contains a standard Windows creation and destruction function, and has other appropriate characteristics required by Windows from time to time). Into the standard Windows application shell, the applications disclosed herein place the code that controls the various threads described above, as well as code for standard Windows message handlers (i.e. code for handling inputs placed on a common message queue for the application), and standard Windows Sockets, or WINSOC, applications (for sending and receiving TCP/IP messages). Included in the decoding thread, there is included the decoder algorithm (which is essentially a mathematical formula that depends on the type of image you are trying to display, e.g., a GIF) and standard Windows library calls to display the decoded images. The main loop of the application can contain code that synchronizes the various threads in the operation. That again is done through a process of using C++ pointers. As each thread executes, it changes either the pointer value or the address of the pointer. As the main loop executes, it checks the value of each pointer to determine the status of the execution of each of the threads and based on a predefined table, executes the appropriate instruction for handling the other threads to result in appropriately synchronized display of different media. Appended to the Windows application at predefined points are the media themselves, which can come from the content providers.

A standard windows executable may contain various elements, including an entry point (header) indicating the length of file, and other data, related to the file, which is found by a standard Windows WIN32 execution file. The length determines the length, e.g., in bytes, of the entire file, which determines in turn the order of the other media packages. The executable may also include an exit point (End of File) and a marker. The Marker identifies the length of the next packet. The various packages in the applications that create 3D visualization environments are then placed between markers, e.g., package, marker, package, marker, and so on. The various elements can be appended in the file in a predetermined order, for execution by the user's computer. Examples of the files that can be appended to the Windows application include 3D animation, audio, video, text, colors to display, fonts, links, hyperlinks, font sizes, window sizes, and any information that can be read by the operating system for its rendering of a multimedia presentation. Hyperlinks can be executed by issuing the embedded URL to the operating system, which, if browser-enabled, can initiate a default browser to handle a URL.

In embodiments, when the user downloads a browser plug-in created by the host, a package can be inserted that consists of a GUID, or global unique identifier, which can be a series of bits (e.g., a 128 bit string) which can be generated by the host server and inserted into the application file. The string can uniquely identify the application file as belonging uniquely to the particular user who is downloading the environment. In a process flow, the user clicks on a browser, the host's server looks for a cookie on the user's browser. If a cookie is found, it is used as the GUID for the file that is going to the user. If no cookie is found, a new GUID is generated. Over time, all files downloaded to a particular user have the same GUID appended to them, and that GUID is appended to all communication from that user to the host system. The host can also pull the GUID out of each click that goes through the host, so that the host can know which user is interacting with it at any time.

The GUIDs can be used to provide a secure authenticated channel. The GUID is used to encrypt messages between the browser plug-in and an e-commerce server. The GUID acts as a private key in a standard public-key/private-key encryption standard, such as DES. For example, in interacting with a 3D visualization environment, a user can click on a hyperlink that opens a web page. The user could then interact with a merchant appearing on the web page without having to undertake other security technology. For example, the system could be used with microbilling providers, which enables central billing for transactions of various providers. This encryption method can be built into the Windows application that establishes the files. The GUID can serve as the key for all transactions for a person using applications created by the toolset.

For a MAC version, the mechanism of delivery of a browser plug-in application may be different. A MAC binary executable application typically cannot be transferred using simple TCP/IP. Therefore, the MAC binary may be encoded into a different form in order to be transported by TCP/IP, in a form known as Binhex, using standard Mac utilities for Binhexing files before sending them over the Internet (e.g., Stuffit). Therefore the host can first create a plug-in application, then use Stuffit or a similar application to transfer it over the Internet. The user's Mac can then unstuff it either as it is downloading or after it is downloaded. The binary file is similar in makeup to the Windows version described above. However, there is no standard function call in the MAC for determining the length of the file; therefore, the host or content provider places an encrypted number in the file to permit the system to know when to execute each particular thread or to know where the end of the entire file is. Similar markers are used to determine the end of the various media packages. Similar algorithms to those of the Windows application are used for synchronizing threads and for decoding audio, video, text and other multimedia, except that MAC uses standard API calls to provide for presentation of the media, rather than Windows calls.

The methods and systems disclosed herein enable a variety of environments in which a variety of business and other methods can be implemented. For example, an online mall can be established, in which users can navigate and shop for various items in a 3D visualization environment. For example, the host may track characteristics of a particular user, such as name, address, gender, demographic segment, geography, zip code, past purchases, activities on the site (such as spending time in certain scenes, undertaking certain activities, and the like). Once these characteristics are tracked, it is possible to construct different 3D visualization environments based on the characteristics of the user. For example, a user who consistently purchases goods and services in a particular color family might be presented with an online store or mall that is represented in those colors. More generally, an avatar may be provided that is based on the user's characteristics.

A wide range of characteristics of the user may be relevant to the 3D environment that will be presented to the user. For example, a 3D visualization can be made of the user, based on that user's physical characteristics, so that goods and services that depend on those characteristics can be presented with an accurate visual presentation. Examples of user characteristics that can be used as a basis for presenting a different visualization include height, weight, size, brand preferences, habits, past spending, hobbies, mouse or keyboard activity, color preferences, eye color, hair color, complexion, birth sign, age, store preferences, expressed likes and dislikes, possessions already owned, geography, zip code, and the like. Thus, profiles can be created so that different, personalized environments are presented, based on the characteristics of the user.

The methods and systems described herein can be used to create Internet sites that are presented with three-dimensional navigation and content capabilities. Thus, a virtual mall can be created, where the user can navigate through the mall as if walking in a real mall; moreover, because data for the environment is stored in a format that can be used for multiple points of view, the same data can produce both the visual representation of the mall, particular stores within the mall, and a three-dimensional site map (e.g., rooftop view) of the mall. The same is true of any other environment, e.g., a virtual store, home, school, park, sports arena, concert arena, vehicle, boat, car, truck, plane, or the like. Moreover, entirely new environments can be created, such as web site where the relevant text, picture, and graphics are presented as if pasted on the interior or exterior of a sphere, cube, or other object, so that navigation through the site can occur by rotating the user's point of view, rather than by clicking on links that display entirely new pages. Such presentations can permit users to have more logical or more entertaining navigation experiences, including realistic immersive experiences wherein the user can navigate to any point of view, not just the limited points of view of particular cameras used to create video or static photographs. Thus, a user can navigate within a web site by panning, tilting, orbiting, rotating, translating, or otherwise moving within an environment, and traditional site information such as text, URLs, pictures, and the like can be positioned however the user desires in three dimensions. For example, a pod can be created in which the user views the interior of the pod, which can contain any type of graphical content.

Environments that benefit particularly from representation via the methods and systems disclosed herein include environments where users wish to visually experience goods, services, or other items. The benefits are especially valuable where the items to be viewed change dynamically, hindering the ability of video or similar technology to represent them, because of the need to take new pictures each time the environment changes. Examples include large retail stores where inventory changes, such as appliance, hardware, or home goods stores, as well as online real estate providers, auto dealers, boat dealers, interior and graphic designers, and a host of others. Environments might include a mall, house, road, furniture store, city, town, or other representation of a real environment, or might include a fantasy environment, such as that provided in a video game.

Objects can be represented within the environments created with the methods and systems described herein. The objects can be easily interchanged, as can textures, colors, and the like. For example, a color and texture can be mapped to any object, and any object can be moved to a different environment, as desired by the host, including in response to user preferences. Objects can include 3D icons (e.g., representing hyperlinks or other logical connections), clothing, models, or the like. For example, an object can be created that resembles the user's physical appearance, so that the user can see a realistic depiction of how another object, such as an item of clothing, will look on the user. Thus, the experience can be geared to the user. Users might be linked to other content based on their preferences, or to their interaction with objects. For example, if a user navigates to a book in an online 3D bookstore, the user could be "hyperjumped" into a three dimensional environment representative of the context of the book, e.g., a medieval environment if the book were about the crusades, or the English countryside if the book were Wuthering Heights. Users can be grouped in affinity groups, so that users with similar preferences can be shown similar 3D environments. Other suitable objects include clothing, appliances, and the like. The methods and systems described herein can be used to present an online tutorial, or how-to manual, for example to assist users in working with an appliance, computer, mechanical object, or the like, by depicting the steps the user needs to take, while simultaneously allowing the user to navigate around the object to see different points of view.

The methods and systems described herein, while providing additional functionality, can also be compatible with existing video- and photograph-based technologies. That is, the data stored in the database and the items drawn by the drawing engine can be based on any sort of data, including video, .gif, .tif, and other formats. Thus, the systems can be integrated with presentations (such as video) that are created for other 3D visualization systems that lack the navigation and progressive refinement capabilities of the present methods and systems.

The preference-based visualization can be created by storing a matrix of user characteristics on the host side in a database, and presenting different scene data based on the characteristics of a particular user.

In an embodiment, a virtual mall may be provided in which the end user views the mall through a view port that appears on the display of the end user's browser. The viewport can consist of a frame of 3D visual material and may be positioned in proximity to one or more other frames that contain various content. For example, the viewport frame may display a virtual store, or a façade for a store, with a sign indicating the name of the store. In that case, an adjacent frame may display other material relevant to the store, such as hyperlinks to a web site of the store, depictions of items sold by the store, advertising displays, or the like. In an embodiment, the frame may appear as a door or window of the store within the view port. For example, the home page of an online store may be depicted as a window of the 3D virtual store that is positioned in the mall.

Navigation through the 3D visualization environments may be made convenient. For example, elements on objects in the environment may be made to appear differently (e.g., glow, blink, etc.), so that when the user clicks on them, the user is moved into close proximity to the objects. Other objects can have different significance. For example, clicking on the door of a virtual store object may cause the user to enter into the home page on another web site belonging to the retailer represented by the store. Also, movement of the mouse or pointer in the 3D environment can result in actions of the environment. For example, moving the mouse near a window on a virtual store might cause the window to display items that can be bought in the store, allowing "no click," virtual window shopping.

The methods and systems disclosed herein can be implemented with electronic commerce transactions systems. For example, a user may, by navigating an environment, click on various objects that represent goods or services the user wishes to procure. Clicking on the objects can cause them to be stored in a virtual shopping cart, and the user can, by taking an action such as clicking on an appropriate object in the environment, purchase the items through the transaction systems. In embodiments, the user can drag objects from a 3D visualization frame into other frames, such as a frame representative of a shopping cart. Thus, a 3D shopping experience can provide the user with an entertaining and convenient shopping experience similar to that of interacting in a real environment.

While the invention has been disclosed in connection with certain preferred embodiments, other embodiments can be envisioned by one of ordinary skill in the art and are encompassed herein.

1. A method for delivery of a 3D visualization environment, comprising:

(a) providing a client application that plugs in to a web browser,

(b) providing a scene database for storing data for a 3D visualization scene, and

(c) providing a server for handling requests from the web browser, wherein in response to requests from the client application via the web browser, the server provides scene data from the scene database and wherein the client application builds a scene for display to the user via the browser.

2. The method of claim 1, wherein the browser displays the scene data in a 3D viewport.

3. The method of claim 2, wherein the viewport is driven by OpenGL or Direct3D.

4. The method of claim 2, wherein the viewport receives input from a user.

5. The method of claim 4, wherein the input from the user is received from a mouse, a keyboard, a touch sensitive screen, a track pad, or a joystick controller.

6. The method of claim 1, wherein the scene database stores data files containing data describing an item capable of graphical representation in a 3D visualization environment.

7. The method of claim 1, wherein the server is provided to the browser by a host using a communication method including TCP/IP, HTTP, FTP or a direct connection.

8. The method of claim 1, wherein the server is provided by a host that is a content provider.

9. The method of claim 1, wherein the server is provided by a host that provides content from a plurality of content providers.

10. The method of claim 2, wherein the viewport allows a scene to be viewed from a plurality of points of view.

11. The method of claim 10, wherein the point of view is selected from the group consisting of the user's point of view and at least one sitemap point of view.

12. The method of claim 11, wherein the sitemap point of view can be displayed in the viewport as an exploded perspective view or an orthogonal view.

13. The method of claim 10, wherein the sitemap point of view can show the user's position in the environment or an anchor point in the environment.

14. The method of claim 1, wherein the environment is provided by prioritized progressive refinement.

15. The method of claim 14, wherein the scene is provided from a compiled scene database that provides a blueprint for the scene from which the client application determines a prioritized list of scene data based on a user's position in the scene, requests from the database the corresponding scene data and updates the scene.

16. The method of claim 15, wherein the scene is initially displayed and then updated using static and dynamic priorities.

17. The method of claim 16, wherein the static priorities are encoded in the scene data and the dynamic priorities are determined by the current user position and orientation on the blueprint and the data requested by the client application is determined by a non-linear weighting of the static and dynamic priorities.

18. The method of claim 1, wherein the scene data is provided in a compiled format.

19. The method of claim 18, where the compiled format of the scene data comprises a header data file and a description data file.

20. The method of claim 20, wherein the header data file includes information about the location of the description data file and instructions on how to request the description data file.

21. The method of claim 14, wherein the data describing items from the user's point of view that are nearest the user in the environment are rapidly retrieved when a user enters a scene and data describing items that are farther away from the user are retrieved when the user is in the scene for a longer period of time.

22. The method of claim 1, wherein the client application provides to the server a global unique identifier.

23. The method of claim 1, wherein the client application requests scene data from more than one server.

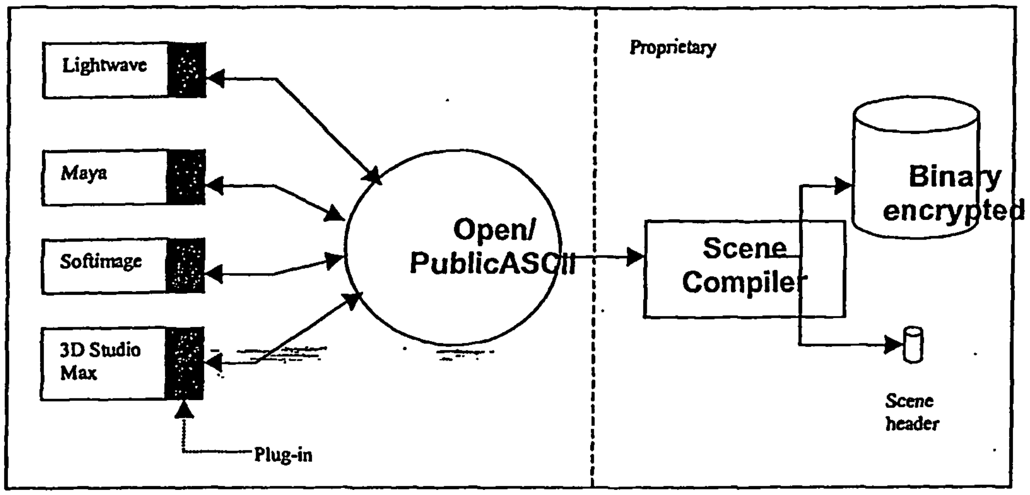

Lightwave
Maya
Softimage
3D Studio Max
Plug-in
Open/ PublicASCII
Proprietary
Scene Compiler
Binary encrypted
Scene header

FIG. 1